

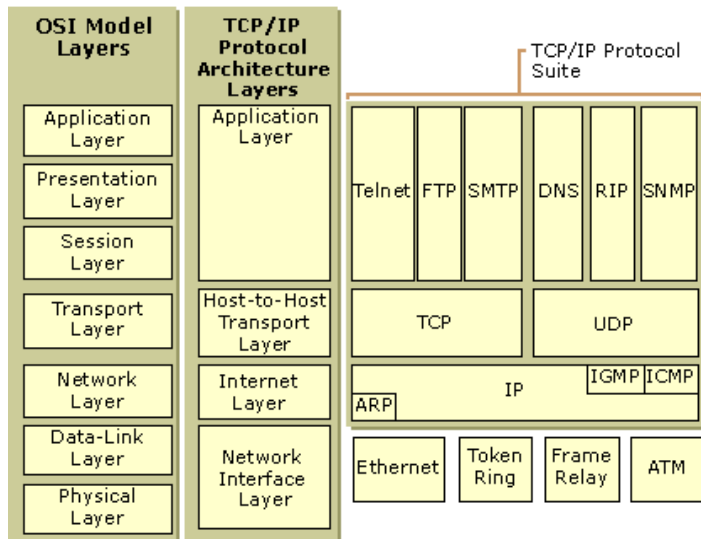
Session V: Up the Stack

Keith Winstein

Assistant Professor of Computer Science
Assistant Professor of Law (by courtesy)
Stanford University



The Stack

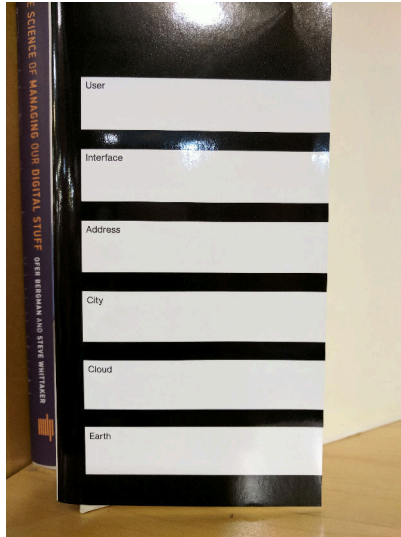




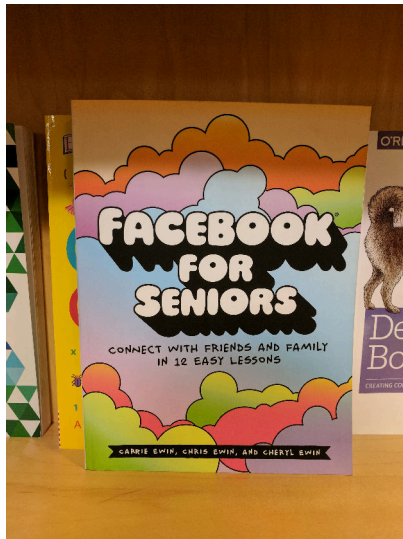
The Stack



The Stack



The Stack



Up the Stack: three papers that push the boundaries

- ▶ **The QUIC Transport Protocol** (Google/ex-Google)

A Web protocol that overlaps and rearranges parts of HTTP, TLS, and TCP.

- ▶ **Neural Adaptive Video Streaming** (MIT)

Neural networks to choose the next “chunk” in an online video (more quality, less quality variation, less startup time, less rebuffering).

- ▶ **Disk|Crypt|Net** (Cambridge, UCL, and Netflix)

Webserver that avoids using memory! Reads straight from disk to cache, encrypts, then sends.

- ▶ **The QUIC Transport Protocol** (Google/ex-Google)

A Web protocol that overlaps and rearranges parts of HTTP, TLS, and TCP.

- ▶ **Neural Adaptive Video Streaming** (MIT)

Neural networks to choose the next “chunk” in an online video (more quality, less quality variation, less startup time, less rebuffering).

- ▶ **Disk|Crypt|Net** (Cambridge, UCL, and Netflix)

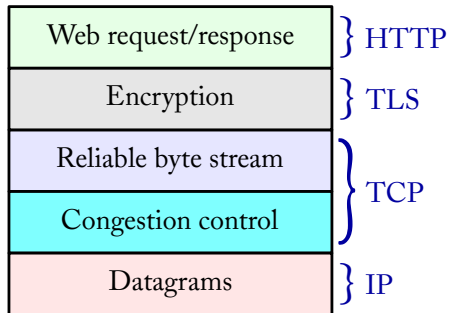
Webserver that avoids using memory! Reads straight from disk to cache, encrypts, then sends.

Back in 1995...

```
<html><body>  
</body></html>
```

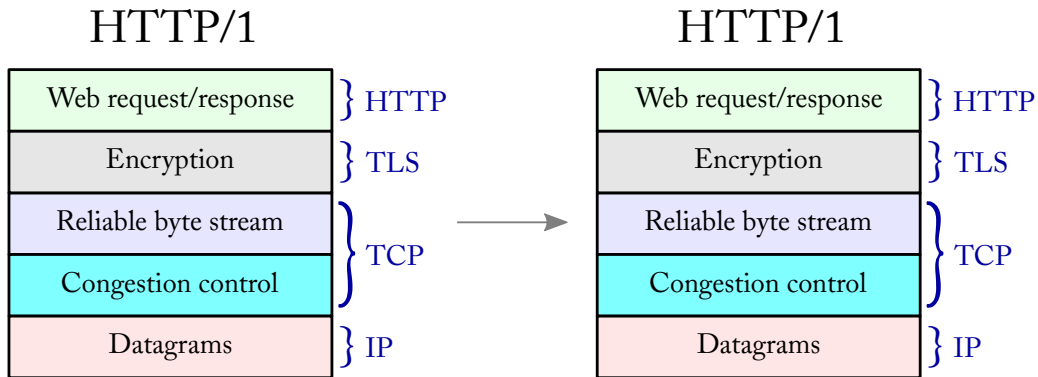
The Web stack in 1995

HTTP/1



(demo)

The Web stack in 1995 (multiple requests)

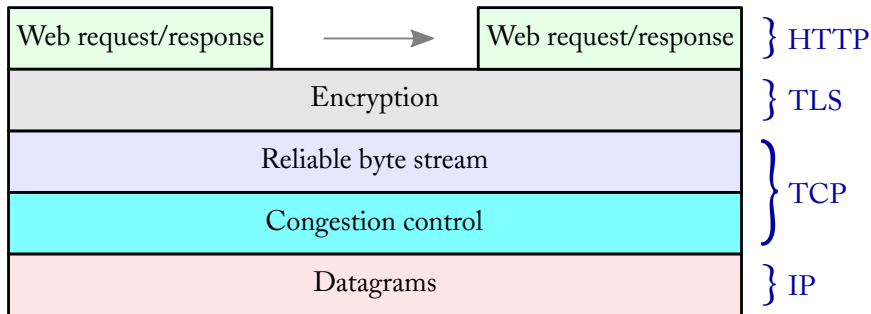


Problem: new connection for each request

Solution: pipeline requests in one connection

The Web stack in 1996

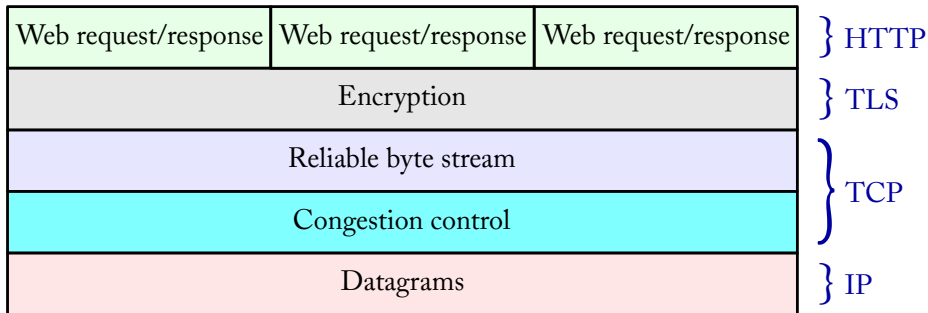
HTTP/1.1



Problem: head-of-line blocking at HTTP layer

Solution: simultaneous HTTP requests

HTTP/2 (SPDY)

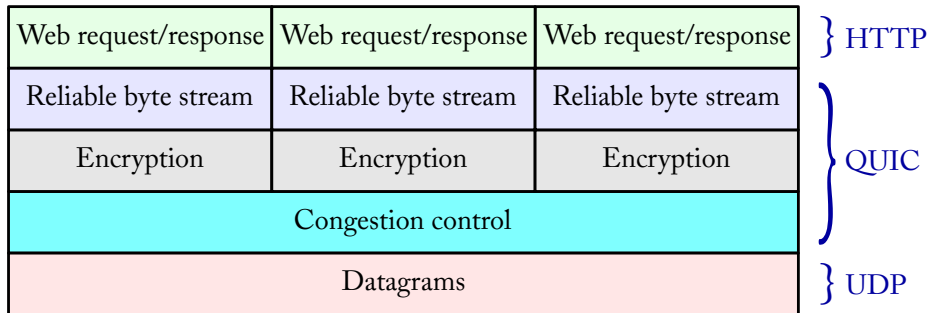


Problem: head-of-line blocking at TCP layer

Solution: separate reliability of different requests

The new Web stack?

QUIC



Problem: head-of-line blocking of datagrams

Solution: low-delay congestion control?
multiple in-network queues?

The QUIC Transport Protocol: Design and Internet-Scale Deployment

Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasnic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, Jeff Bailey, Jeremy Dorfman, Jim Roskind, Joanna Kulik, Patrik Westin, Raman Tenneti, Robbie Shade, Ryan Hamilton, Victor Vasiliev, Wan-Teh Chang, Zhongyi Shi *

Google

quic-sigcomm@google.com

ABSTRACT

We present our experience with QUIC, an encrypted, multiplexed, and low-latency transport protocol designed from the ground up to improve transport performance for HTTPS traffic and to enable rapid deployment and continued evolution of transport mechanisms. QUIC has been globally deployed at Google on thousands of servers and is used to serve traffic to a range of clients including a widely-used web browser (Chrome) and a popular mobile video streaming app (YouTube). We estimate that 7% of Internet traffic is now QUIC. We describe our motivations for developing a new transport, the principles that guided our design, the Internet-scale process that we used to perform iterative experiments on QUIC, performance improvements seen by our various services, and our experience deploying QUIC globally. We also share lessons about transport design and the Internet ecosystem that we learned from our deployment.

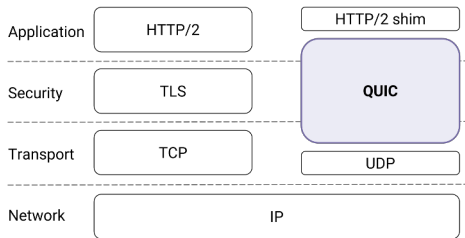


Figure 1: QUIC in the traditional HTTPS stack.

Some key results

		% latency reduction by percentile						
		Lower latency				Higher latency		
	Mean	1%	5%	10%	50%	90%	95%	99%
Search								
Desktop	8.0	0.4	1.3	1.4	1.5	5.8	10.3	16.7
Mobile	3.6	-0.6	-0.3	0.3	0.5	4.5	8.8	14.3
Video								
Desktop	8.0	1.2	3.1	3.3	4.6	8.4	9.0	10.6
Mobile	5.3	0.0	0.6	0.5	1.2	4.4	5.8	7.5

Table 1: Percent reduction in global Search and Video Latency for users in QUIC_g, at the mean and at specific percentiles. A 16.7% reduction at the 99th percentile indicates that the 99th percentile latency for QUIC_g is 16.7% lower than the 99th percentile latency for TCP_g.

		% rebuffer rate reduction by percentile				
		Fewer rebufferers			More rebufferers	
	Mean	< 93%	93%	94 %	95%	99%
Desktop	18.0	*	100.0	70.4	60.0	18.5
Mobile	15.3	*	*	100.0	52.7	8.7

Table 2: Percent reduction in global Video Rebuffer Rate for users in QUIC_g at the mean and at specific percentiles. An 18.5% reduction at the 99th percentile indicates that the 99th percentile rebuffer rate for QUIC_g is 18.5% lower than the 99th percentile rate for TCP_g. An * indicates that neither QUIC_g nor TCP_g have rebufferers at that percentile.

Some key results (continued)

Country	Mean Min RTT (ms)	Mean TCP Rtx %	% Reduction in Search Latency		% Reduction in Rebuffer Rate	
			Desktop	Mobile	Desktop	Mobile
South Korea	38	1	1.3	1.1	0.0	10.1
USA	50	2	3.4	2.0	4.1	12.9
India	188	8	13.2	5.5	22.1	20.2

Table 3: Network characteristics of selected countries and the changes to mean Search Latency and mean Video Rebuffer Rate for users in QUIC_g.

Possible areas to ponder

- ▶ **Locus of gains**

The benefit of QUIC (compared with kernel TCP) varies by RTT, geography, and desktop/mobile. Do gains also vary based on kernel TCP? Maybe some kernels are lousier than others!

- ▶ **Reproducibility**

Google has not published the “real” (GFE) QUIC server, only a “toy” implementation. Will outsiders get the same gains based on the code that has been released or their own implementations?

- ▶ **What are next steps after QUIC?**

Switchable congestion control in user space allows rapid experimentation!

- ▶ **The QUIC Transport Protocol** (Google/ex-Google)

A Web protocol that overlaps and rearranges parts of HTTP, TLS, and TCP.

- ▶ **Neural Adaptive Video Streaming** (MIT)

Neural networks to choose the next “chunk” in an online video (more quality, less quality variation, less startup time, less rebuffering).

- ▶ **Disk|Crypt|Net** (Cambridge, UCL, and Netflix)

Webserver that avoids using memory! Reads straight from disk to cache, encrypts, then sends.

The problem: **adaptive** streaming of **dead** bits

youtube gangnam style


Secure <https://www.google.com/search?q=youtube+gangnam+style&soq=youtube+gangnam+style&q=chrome..69157015.3008j0j7&client=ubuntu&sourceid=chrome&ie=UTF-8>

Google

youtube gangnam style

All Videos News Shopping Books More Settings Tools

About 2,150,000 results (0.87 seconds)



PSY - GANGNAM STYLE(강남스타일) M/V - YouTube

<https://www.youtube.com/watch?v=9bZkp7q19f0>

Lyrics

오빤 강남스타일
강남스타일
낮에는 따사로운 인간적인 여자

How streaming works

- ▶ Video encoder compresses **multiple** versions of video at different bitrates.
- ▶ Encoding is split into independent **chunks** (2–5 seconds each).
- ▶ Chunk boundaries are synchronized.
- ▶ At runtime, players fetch the best version of the video that their network connection allows.
- ▶ If network changes (for worse or better), change quality of future chunks.

Typical implementation

- ▶ Server = webserver/CDN
 - ▶ Player = JavaScript (e.g. `dash.js`, YouTube JavaScript)
 - ▶ Protocol = HTTP
-
- ▶ This is called “Dynamic Adaptive Streaming over HTTP.”

Goals in video streaming

- ▶ **Maximize:** Video quality (compared with original)
- ▶ **Minimize:** Time before video starts (to accumulate buffer)
- ▶ **Minimize:** Number of pauses (to accumulate more buffer)
- ▶ **Minimize:** Amount of time spent rebuffering
- ▶ **Minimize:** Variability in video quality

Other possible goals, but generally out-of-scope in academic work:

- ▶ Amount of time users spend watching videos
- ▶ Advertising revenue for hosting company
- ▶ Learning, information retention, edification
- ▶ Preserving a functioning democracy

Algorithms for video streaming

- ▶ **“Rate-based”**

- ▶ **“Buffer-based”**

(Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, Mark Watson, SIGCOMM 2014)

- ▶ **Model-predictive control**

(Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, Bruno Sinopoli, SIGCOMM 2015)

- ▶ **Neural networks**

(SIGCOMM 2017)

Neural Adaptive Video Streaming with Pensieve

Hongzi Mao, Ravi Netravali, Mohammad Alizadeh
MIT Computer Science and Artificial Intelligence Laboratory
{hongzi,ravinet,alizadeh}@mit.edu

ABSTRACT

Client-side video players employ adaptive bitrate (ABR) algorithms to optimize user quality of experience (QoE). Despite the abundance of recently proposed schemes, state-of-the-art ABR algorithms suffer from a key limitation: they use fixed control rules based on simplified or inaccurate models of the deployment environment. As a result, existing schemes inevitably fail to achieve optimal performance across a broad set of network conditions and QoE objectives.

We propose Pensieve, a system that generates ABR algorithms using reinforcement learning (RL). Pensieve trains a neural network model that selects bitrates for future video chunks based on observations collected by client video players. Pensieve does not rely on pre-programmed models or assumptions about the environment. Instead, it learns to make ABR decisions solely through observations of the resulting performance of past decisions. As a result, Pensieve automatically learns ABR algorithms that adapt to a wide range of environments and QoE metrics. We compare Pensieve to state-of-the-art ABR algorithms using trace-driven and real world experiments spanning a wide variety of network conditions, QoE metrics, and video properties. In all considered scenarios, Pensieve outperforms the state-of-the-art, achieving higher user quality of experience (QoE) and lower buffering than existing algorithms.

content providers [12, 25]. Nevertheless, content providers continue to struggle with delivering high-quality video to their viewers.

Adaptive bitrate (ABR) algorithms are the primary tool that content providers use to optimize video quality. These algorithms run on client-side video players and dynamically choose a bitrate for each video *chunk* (e.g., 4-second block). ABR algorithms make bitrate decisions based on various observations such as the estimated network throughput and playback buffer occupancy. Their goal is to maximize the user’s QoE by adapting the video bitrate to the underlying network conditions. However, selecting the right bitrate can be very challenging due to (1) the variability of network throughput [18, 42, 49, 52, 53]; (2) the conflicting video QoE requirements (high bitrate, minimal rebuffering, smoothness, etc.); (3) the cascading effects of bitrate decisions (e.g., selecting a high bitrate may drain the playback buffer to a dangerous level and cause rebuffering in the future); and (4) the coarse-grained nature of ABR decisions. We elaborate on these challenges in §2.

The majority of existing ABR algorithms (§7) develop fixed control rules for making bitrate decisions based on estimated network throughput (“rate-based” algorithms [21, 42]), playback buffer size (“buffer-based” schemes [19, 41]), or a combination of the two

How it works

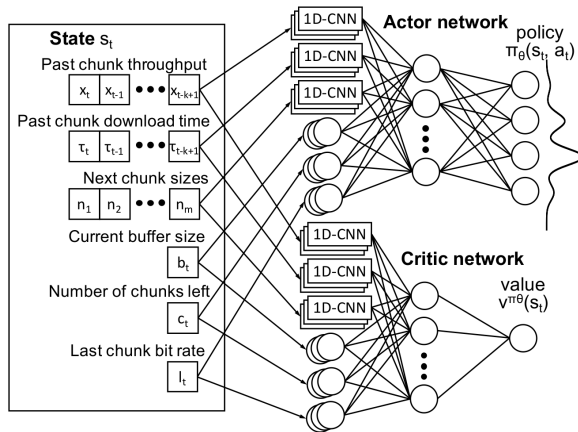


Figure 5: The Actor-Critic algorithm that Pensieve uses to generate ABR policies (described in §4.4).

Key results

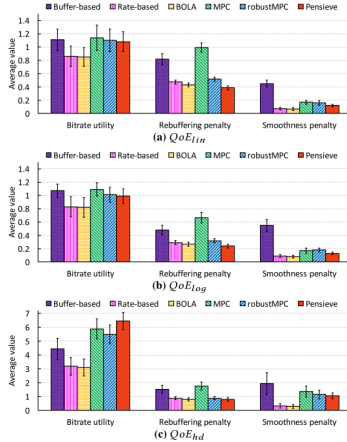


Figure 10: Comparing Pensieve with existing ABR algorithms by analyzing their performance on the individual components in the general QoE definition (Equation 6). Results consider both the broadband and HSDPA networks. Error bars span \pm one standard deviation from the average.

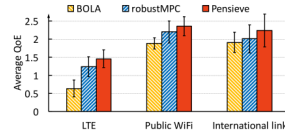


Figure 11: Comparing Pensieve with existing ABR algorithms in the wild. Results are for the QoE_{lin} metric and were collected on the Verizon LTE cellular network, a public WiFi network, and the wide area network between Shanghai and Boston. Bars list averages and error bars span \pm one standard deviation from the average.

Possible areas to ponder

- ▶ **Classical vs. neural**

Where do the gains come from? Would it be possible to take insights from this paper to make classical approaches (e.g. MPC) do as well?

- ▶ **Reproducibility**

The paper evaluates against these authors' *own* implementation of the MPC competition. Do the results hold up against real competitors? Or against YouTube?

- ▶ **Interpretation**

Is there a way to interpret the QoE functions? Does it make sense to *maximize* bitrate?

- ▶ **Next steps**

Are large gains available if getting rid of HTTP/TCP, or the independence of video “chunks”?

- ▶ **The QUIC Transport Protocol** (Google/ex-Google)

A Web protocol that overlaps and rearranges parts of HTTP, TLS, and TCP.

- ▶ **Neural Adaptive Video Streaming** (MIT)

Neural networks to choose the next “chunk” in an online video (more quality, less quality variation, less startup time, less rebuffering).

- ▶ **Disk|Crypt|Net** (Cambridge, UCL, and Netflix)

Webserver that avoids using memory! Reads straight from disk to cache, encrypts, then sends.

TLS webserving is what it's all about

- ▶ **Most** consumer Internet traffic is HTTP fetches of dead bits for streaming video!
- ▶ Commercial need to serve these files efficiently off disk
- ▶ Can buy a PC with 80 Gbps of NICs; how many cores required?
- ▶ TLS webserving at 80 Gbps is not so easy!

The problem: memory bandwidth

- ▶ **User** requests video chunk
- ▶ **Webserver** reads file from disk to RAM
- ▶ **TLS** reads from RAM, encrypts, writes to RAM
- ▶ **TCP** reads from RAM, copies to NIC

Problem: need more than 8 cores

Disk|Crypt|Net: rethinking the stack for high-performance video streaming

Ilias Marinos
University of Cambridge

Robert N.M. Watson
University of Cambridge

Mark Handley
University College London

Randall R. Stewart
Netflix Inc.

ABSTRACT

Conventional operating systems used for video streaming employ an in-memory disk buffer cache to mask the high latency and low throughput of disks. However, data from Netflix servers show that this cache has a low hit rate, so does little to improve throughput. Latency is not the problem it once was either, due to PCIe-attached flash storage. With memory bandwidth increasingly becoming a bottleneck for video servers, especially when end-to-end encryption is considered, we revisit the interaction between storage and networking for video streaming servers in pursuit of higher performance.

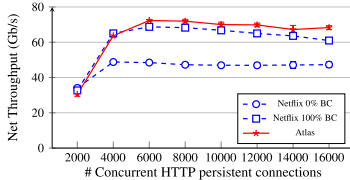
We show how to build high-performance userspace network services that saturate existing hardware while serving data directly from disks, with no need for a traditional disk buffer cache. Employing netmap, and developing a new *diskmap* service, which provides safe high-performance userspace direct I/O access to NVMe devices, we amortize system overheads by utilizing efficient batching of outstanding I/O requests, process-to-completion, and zerocopy operation. We demonstrate how a buffer-cache-free design is not only practical, but required in order to achieve efficient use of memory bandwidth on contemporary microarchitectures. Minimizing latency between DMA and CPU access by integrating storage and TCP control loops

1 INTRODUCTION

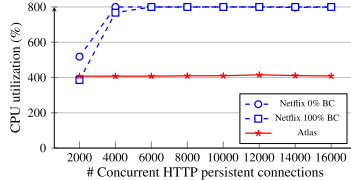
More than 50% of Internet traffic is now video streamed from services such as Netflix. How well suited are conventional operating systems to serving such content? In principle, this is an application that might be well served by off-the shelf solutions. Video streaming involves long-lived TCP connections, with popular content served directly from the kernel disk buffer cache using the OS `sendfile` primitive, so few context switches are required. The TCP stack itself has been well tuned over the years, so this must be close to a best-case scenario for commodity operating systems.

Despite this, Netflix has recently committed a number of significant changes to FreeBSD aimed at improving streaming from their video servers. Perhaps current operating systems are not achieving close to the capabilities of the underlying hardware after all?

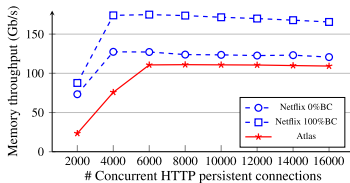
Previous work[18] has shown that a specialized stack can greatly outperform commodity operating systems for short web downloads of static content served entirely from memory. The main problem faced by the conventional stack for this workload was context switching between the kernel and OS to accept new connections; this solution achieves high performance by using a zero-copy architecture closely coupling the HTTP server and the TCP/IP stack in userspace.



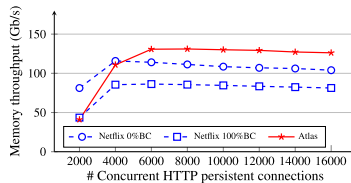
(a) Network throughput (Error bars indicate the 95% CI)



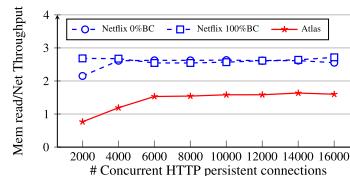
(b) CPU utilization (Average)



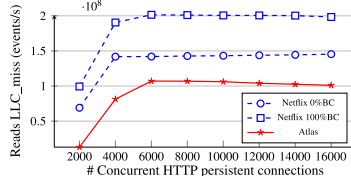
(c) Memory READ



(d) Memory WRITE



(e) Memory READ-Network Throughput Ratio



(f) CPU reads served from DRAM due to LLC misses

Figure 13: Encrypted performance, Netflix vs. Atlas, zero and 100% Buffer Cache (BC) ratios.

Possible areas to ponder

- ▶ **Great results, but why so far from goal?**

Hope was 0% use of RAM, but reality is 150%! Why?

- ▶ **What are generalizable lessons?**

If RAM is the enemy, do operating systems need to be fundamentally refactored? How?